Artificial Intelligence Project--RLE and MIT Computation Center Memo 16--A Question-Answering Routine[1, 2]

by

Anthony Valiant Phillips

A program has been written in the LISP programming language to answer English-language questions by consulting an English-language text. The program can handle questions about the subject, verb, place and time of simple sentences. The program proceeds in two steps. In the first, the machine analyzes the question and the sentences of the text, and puts them into a form in which they can be compared. For this analysis the machine must have as input a dictionary of part-of-speech tags, and a set of rules, analogous to phrase-structure rules, according to which it will organize the sentences. This analysis organizes the sentences into noun-phrases, verbs, and prepositional phrases. The machine then picks from the sentence a subject, a verb, an object, and prepositional phrases relating to place and time. This is the "canonical form" of the sentence.

The next part of the program compares the question with each of the sentences in the text. Those that match, i.e. contain the information the question is asking for, are stored and the answer is made up from them. If none are found, an appropriate negative answer is given.

This routine has been debugged and has run successfully.

Introduction

The original aim of this research was a routine that could pass a reading-comprehension text suitable for six or seven-year-old children, and even though we had to lower our sights, this still gives the best idea of the purpose of this paper. By the time a child has learnt to read, he is already capable of feats of data analysis that seem very difficult, if not impossible, to duplicate on a machine. The child has picked up the key to

---

[1]The author wrote this paper while working with the Mechanical Translation group in RLE, supported in part by the National Science Foundation and in part by the U.S. Army (Signal Corps), the U.S. Air Force (Office of Scientific Research, Air Research, and Development Command) and the U.S. Navy (Office of Naval Research). In addition, this work was done in part at the MIT Computation Center, Cambridge, Massachusetts.

[2]This paper was submitted to the Department of Mathematics on May 21, 1960 in partial fulfillment of the requirements for the degree of Master of Science. Thesis Supervisor for this paper was John McCarthy, Assistant Professor of Communication Sciences, Department of Electrical Engineering, MIT, Cambridge, Massachusetts.

syntactic analysis, and has begun to assimilate the huge mass of common knowledge that is taken for granted in human communication. He is also becoming initiated, although slowly*, to the habit of reasoning, and is learning to handle logical connectives and quantifiers. These are all directions this paper did not explore.

The idea behind the answering process that will be described in the following pages is very simple. We assume that each question asks for a piece of information, and that pieces of information are contained in the text, one to a sentence. The answering routine consists then in trying to match the question against successive sentences of the text to see if any one of them contains the information asked for.

The routine first performs an analysis of the question and of each sentence of the text, and edits them into a form in which they can be compared. It then performs the comparison, and prints out appropriate answers. The following examples are taken from actual machine runs, not using the three functions mentioned above.

Examples: text: ((AT SCHOOL JOHNNY MEETS THE TEACHER)
(THE TEACHER READS BOOKS IN THE CLASSROOM))

question: (WHERE DOES THE TEACHER READ BOOKS)

answer: (((IN THE CLASSROOM) (THE TEACHER READS BOOKS IN THE CLASSROOM)))

text: (JOHNNY GOES TO SCHOOL IN THE MORNING)

question: (WHOM DOES JOHNNY MEET)

answer: (THE ORACLE DOES NOT KNOW)

Additional input is required for the sentence analysis: in the examples above, the machine had also been fed three lists. (These are part of the "a-list".) The first, denoted by GRAMM1, was a part-of-speech "dictionary" of the form ((M THE) (N TEACHER) (V READS) (N BOOKS) (P IN) (NPL CLASSROOM) (P AT) (N JOHNNY) (V MEETS) (V GOES) (P TO) (NTI MORNING) (NPL SCHOOL) (AUX1 DOES) (AUX1 DO) (V READ) (V SIT) (Q WHERE) (Q WHAT) (Q WHOM) (N MARY)).

---

*See Piaget and Inhelder, La Genese des Structures Logiques Elementaires, Delachaux et Niestle, Neuchatel, 1959.

The second list, GRAMM2, is a set of "grammar rules" which the machine uses to analyze the sentences and the question. For the examples above, the GRAMM2 list was (((M NPL) NPL) ((M N) N) ((P NPL) PNPL) ((P NTI) PNTI) ((M NTI) NTI) ((P NTI) PNTI)). The third list, referred to as Z, is of the form (((DOES GO) GOES) ((DOES DO) DOES) ((DOES READ) READS)). These three lists, the sentence and the text are processed by the program described in the following pages. and printed out in A

Work of a similar nature is being done at the Lincoln Laboratory of MIT in B. F. Green's group. They are interested in answering questions pertaining to a table, in this case to the table of 1958 baseball scores. Their work should be nearly finished, but up to now little information has been released about the details of their program. (See the Lincoln Laboratory QPR for Division 5, December 15, 1959.) I believe that they have had to analyze special segments of the syntax more thoroughly ("exactly three..", "more than", etc.) becuase of the nature of their text and the questions expected about it.

Our exposition will proceed in the following manner: first, to explain how our answering process works, we shall describe a simple example of a question-answering routine, answer1. This routine was written at the beginning of this study, and was debugged and will run successfully as it is presented here. It answers simple-minded questions about a table, but its basic mechanism is the same as that of the large program. After that we will present a description of the main program, followed by a more detailed examination of each of its subfunctions.

A Simple Example: The program answer1

amswer1 takes as input a question and a table. The table tells whether or not a relation R exists between any two letters of the alphabet.* The table is arranged as follows:

(A R B B R C A R C ...)

The questions that may be asked are:

Does A R B?

What R B?

What does B R? , where A, B may be any two letters.

The program described searches the "text" for an answer to the question. There is no provision for recording two or more

---

*It will be seen below that S cannot be used.

answers if they should occur. Thus if A R C and A R D both appear in the table, the answer to the question "What does A R?" will be "A R C" if this entry happens to come first in the table. The answer will be "A R Nothing" if there is no entry of the form A R ... .

answer1 works as follows: It first performs qanal of the question, translating it into a form where it can be easily matched against the table:

qanal changes	"Does A R B?"	into	A R B
	"What R B?"	into	S R B
	"What does B R?"	into	BRR S

It then performs test to compare the analyzed question with the text. test uses testa to match the question against each successive table entry. If no matches are recorded, the subroutine neg is performed, which takes the question as input and supplies the correct negative answer: "A does not R B", "Nothing R B" or "B R nothing".

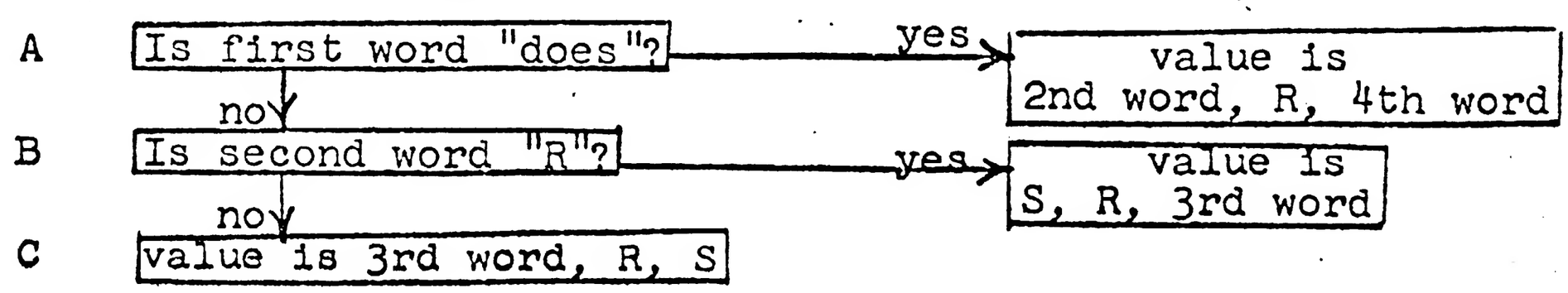
testa matches the question against an entry by using same to compare the "words of the question with the corresponding "words" of the table entry. same will register a match if the two elements it is comparing are identical, or if the question-element is "S", the symbol we are using for "what". This is expressed in the following table:

element of table-entry

element of question	A	B
A	MATCH	NO
S	MATCH	MATCH

The LISP program for answer1*

We will describe qanal in more detail than the rest of the program: We want qanal to operate as follows:



*for a description of the LISP language see the paper by John McCarthy entitled "Recursive Functions of Symbolic Expressions and Their Computation by Machine" in Communications of the ACM, Vol. 3, No. 4, April, 1960.

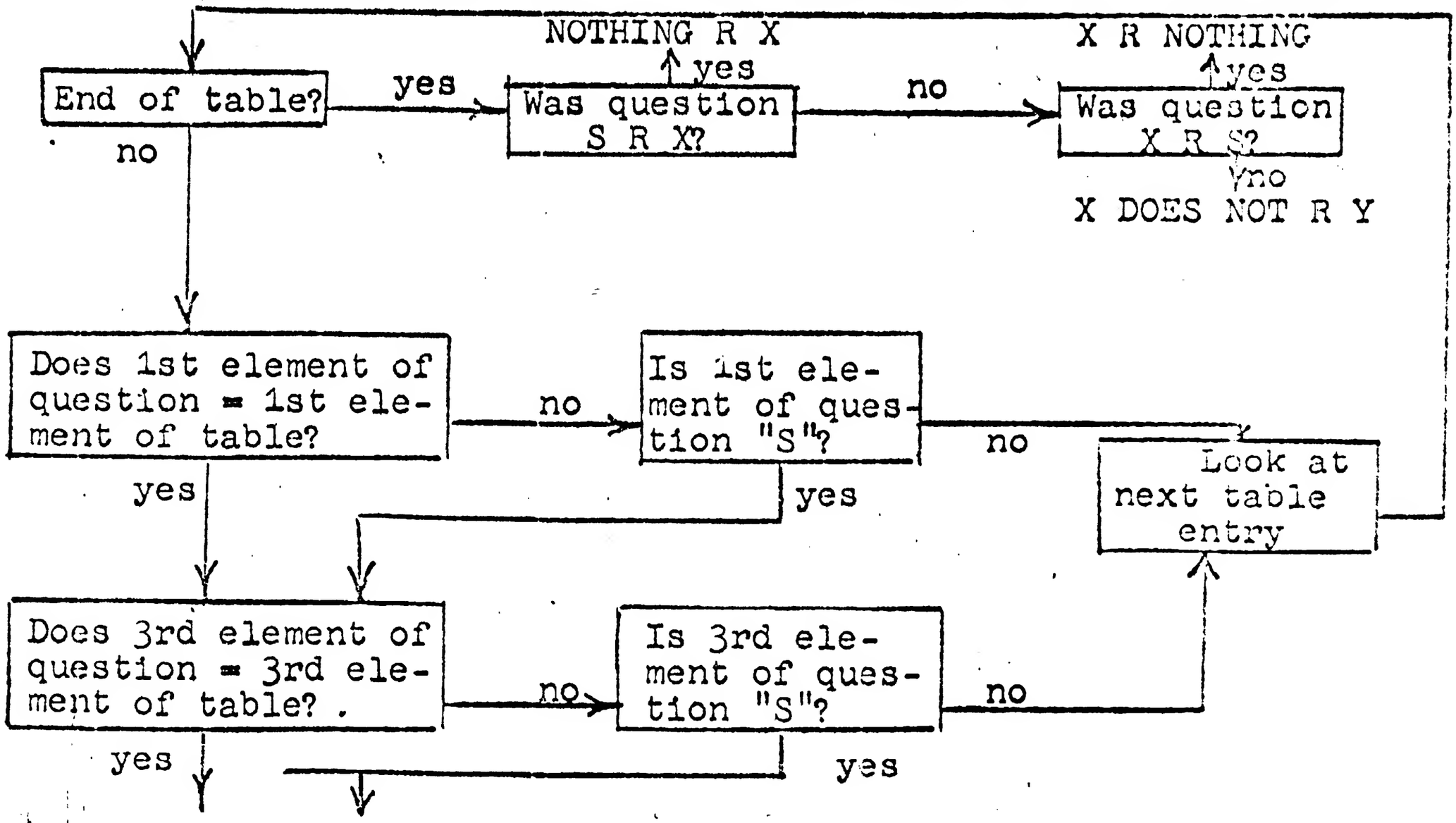
We then express A as:

```
car[X] = quote[DOES] → list[cadr[X],quote[R],cadddr[X]]
```

B and C are expressed similarly, and we write:

```
qanal[X] =[car[X] = quote[DOES] → list[cadr[X];quote[R];
cadddr[X]];cadr[X] = quote[R] → list[quote[S];quote[R];
caddr[X]];T → list[caddr[X];quote[R];quote[S]]]
```

answer1 and its other subfunctions can be described in a similar manner. We want them to act according to the flowchart:



Answer is: 1st element of table, "R", 3rd element of table.

LISP definitions:

```
answer1[X;Y] = test[qanal[X];Y]
test[X;Y] = [null[Y] → neg[X];T → testa[X;Y]]
testa[X;Y] = [same[car[X];car[Y]] → [same[caddr[X];
                caddr[Y]] → list[car[Y];quote[R];caddr[Y]];
                T → test[X;cdddr[Y]]];T → test[X;cdddr[Y]]]
same[X;Y] = [X = Y → T;X = quote[S] → T;T → F]
```

```
neg[X] = [car[X] = quote[S] → list[quote[NOTHING];quote[R];
caddr[X]];caddr[X] = quote[S] → list[car[X];quote[R];
quote[NOTHING]];T → list[car[X];quote[DOES];quote[NOT];
quote[R];caddr[X]]]
```

We now wish to apply a similar matching procedure to an English-language question and text. The principal task will be to put the text in the form of a table, and the question in a form suitable for comparison with the "table" entries. This work will be accomplished by the functions lex, parse, revert, edit, order which make up the "syntactic analysis" portion of the routine. The matching will be done by answer, ques and anal. These are all subfunctions of the main function, oracle, whose arguments are a question and a text, and whose value is the answer or answers to the question found in the text.

The function oracle:

I. The Syntactic Analysis

It was outside the scope of this project to embark on a systematic and correct analysis of even a very restricted class of English sentences. Accordingly a number of artificial assumptions were made about the sentences to be analyzed. The most important one is that no ambiguity will be encountered at any level in the analysis. As we assign part-of-speech tags to the words in the analyzed sentences, this assumption means, for instance, that a given word can be only one part of speech. There is no provision, in the analysis program which will be described here, for backtracking or for maintaining two or more distinct hypotheses about the structure of the sentence. The analysis proceeds in two steps.

The first step, embodied in the functions lex and parse, goes through the sentence looking for nouns and grouping with each one the adjectival structure that belongs to it. If a noun is preceded by a preposition, a prepositional phrase is formed.

The sentence is presented to this analysis in the following form: each word is tagged with a part-of-speech symbol. Here the symbols used are N (noun), M (noun-modifier), V (verb), NPL (place-noun, e.g. playground), NTI (time-noun, e.g. noon)*, P (preposition), and AUX1 for forms of the verb to do which play a privileged role. (This format is the result of applying lex to the sentence and the part-of-speech "dictionary".) Also

---

*this distinction is necessary to distinguish "in the garden" (answer to"where?") from "in the afternoon" (answer to "when?").

part of the input is a set of rules for forming noun-phrases and prepositional phrases*, e.g.

M + N = N

M + NTI = NTI

P + NTI = PNTI, etc.

There are also rules of the form

V + V = V

for grouping an auxiliary-participle combination into a single element. ("M + N = N" means that when a word or group of words tagged M is followed by a word or group of words tagged N, the two groups should be consolidated into one, tagged N.)

An example of theaaction of lex and parse:

sentence (JOHNNY SEES THE DOG IN THE GARDEN)

"dictionary" ((N JOHNNY) (N DOG) (NPL GARDEN) (V SEES) (P IN) (M THE))

rules (((M N)NN) ((M NPL) NPL) ((P NPL) NPL))

The three inputs are shown in the format in which they are acceptable to the routine. ((M N) N) is a representation of M + N = N. The result of the analysis is:

((N JOHNNY) (V SEES) (N THE DOG) (PNPL IN THE GARDEN)).

Up to this point the analysis is in some sense independent of the routine since, if the ambiguity clause is respected, "dictionary" and grammar rules are part of the input. This concludes the first step.

The second step: It was decided that the routine would be written only to handle questions dealing with the subject, object, place-phrase and time-phrase of a sentence. This implies immediately, for example, that only simple sentences will be considered. The function order does this editing. Its result is a list of five elements, subject, verb, object, place and time, in that order. Anything else in the sentence will be discarded; missing elements, except subject and verb whose presence is required, have their place marked by an (S). We give an example of the combined action of lex, parse and order; these three functions are responsible for the analysis of the sentence.

---

*These rules are inspired by Noam Chomsky's phrase-structure rules. (See his Syntactic Structures, Mouton and Co., 'S-Gravenhage, 1957, p. 26)

sentence (IN THE GARDEN THE LITTLE BOY GAVE THE DOG A BONE)

dictionary ((NPL GARDEN) (N BOY) (N DOG) (N BONE) (V GAVE)
(M A) (M THE) (M LITTLE) (P IN))

rules (((M N) N) ((M NPL) NPL) ((P NPL) PNPL))

result: ((THE LITTLE BOY) (GAVE) (THE DOG) (IN THE GARDEN) (S))

The routine cannot handle double objects.

The analysis of the question uses the same functions: lex, parse and order. It also needs revert to cope with the inversion characteristic of the question-sentence in English, and edit to take care of question-words. revert requires as part of its input a table showing how to combine the auxiliary to do with participles, e.g.

DOES + GIVE = GIVES

DID + GO = WENT, etc.

We give examples of the application of lex, parse, revert, edit and order, in that order, to interrogative sentences:

sentences (WHERE DID THE TEACHER GO)

(HAS JOHNNY COME HOME)

dictionary ((Q WHERE)* (AUX1 DID) (M THE) (N TEACHER) (V GO) (V HAS)
(N JOHNNY) (V COME) (PNPL HOME))

rules (((M N) N))

table (((DID GO) WENT))

results ((THE TEACHER) (WENT) (S) (W)* (S))

((JOHNNY) (HAS COME) (S) (HOME) (S))

II. The Matching

The sentence and the question under consideration have now been edited into a canonical form in which they can be compared. The comparison proceeds as follows: the first block in the "canonical form" of the question is matched against the first in that of the sentence, and so forth. If all agree, question and sentence are said to match.

The blocks are compared as in the table below, where T appears at the conjunction of two entries if a match would be recorded, and F if not.

---

*Q is a part-of-speech tag for question-words; (W) in fourth position represents WHERE.

Block n of the sentence is:

Block n of the question is:

	(S)	(A B)	(C)
(S)	T	T	T
(W)	F	T	T
(B)	F	TT	F

For example, if block n of the question is (THE OLD YELLOW DOG) and block n of the sentence is (THE UGLY OLD YELLOW DOG), these blocks will match. If block n of the sentence were (THE DOG) or (THE YELLOW CANARY), there would be no match. If block n of the question is empty, it will match whatever block n of the question contains. If block n of the question is a question-word, (e.g. a (W) is the first block representing a "who"), it will match with block n of the sentence as long as the latter block is not empty.

The functions of oracle

lex[X;Y] has as arguments a list of words, the sentence, and a list of pairs, the "dictionary". Its value is the list of the dictionary entries corresponding to the words in the sentence. lex has a subfunction lookup.

llookup[X;Y] has as arguments a word and a list of pairs. Its value is the pair whose second element is the word in question. Its value is ERROR if there is no such pair.

Examples:

X = JOHNNY

Y = ((N MARY) (N JOHNNY) (V SITS), etc.)

lookup[X;Y] = (N JOHNNY)

X = (JOHNNY GOES TO SCHOOL)

Y = ((P TO) (N MARY) (V GOES) (N JOHNNY) (NPL SCHOOL))

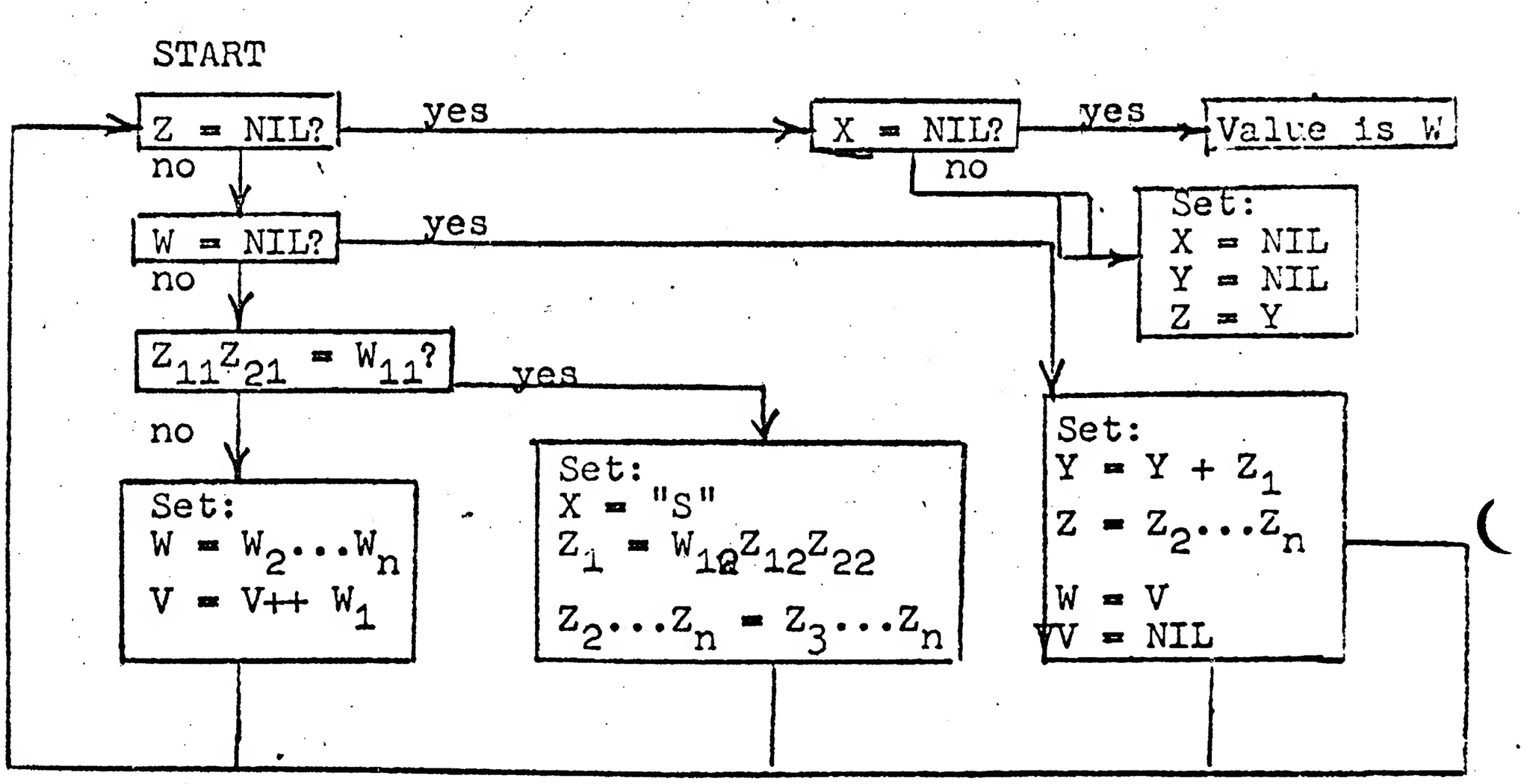
lex[X;Y] = ((N JOHNNY) (V GOES) (P TO) (NPL SCHOOL))

LISP definitions:

```
lookup[X;Y] = [null[Y] → ERROR;T → [X = cadar[Y] → car[Y];
                                    T → lookup[X;cdr[Y]]]]

lex[X;Y] = [null[X] → NIL;T → cons[lookup[car[X];Y];
                                   lex[cdr[X];Y]]]
```

parse[X;Y] has as arguments a list of pairs (the result of lex of the sentence) and another list of pairs, the rules. Elements of this second list are of the form ((A B) C), corresponding to rules of the form A + B = C. The work in parse is done by the subfunction glob.

glob[X;Y;Z;W;V]: X is a marker which tells whether or not any rule has been applied in the last pass through the sentence. Y contains the sentence elements that have already been examined in the current pass; Z contains the rest of the sentence; W contains the rules that have not yet been tried in the analysis of the first two elements of Z; V contains those that have been tried and have failed. The action of glob is described in the flowchart below, where Z_1 represents the first element of Z (car[Z]) and Z_{12} represents the second element of Z_1.



Example:

X = ((M THE) (M NASTY) (M OLD) (N MAN))

Y = (((M N) N))

parse[X;Y] = ((N THE NASTY OLD MAN))

LISP definitions:

```
glob[X;Y;Z;W;V] = [null[Z] → [null[X] → Y;T →
glob[NIL;NIL;Y;W;V]];null[W] → glob[X;nconc[Y;cons[car[Z]$
NIL]];cdr[Z];V;NIL];list[caar[Z];caadr[Z]] = caar[W] →
glob[quote[S];Y;cons[append[cdar[W];append[cdar[Z];cdadr[Z]]];
cddr[Z]];W;V];T → glob[X;Y;Z;cdr[W];nconc[V;cons[car[W];
NIL]]]]

parse[X;Y] = glob[quote[S];X;NIL;Y;NIL]
```

order[X] has as its argument a list of pairs, the value of parse[lex[sentence;dictionary];rules]. Its value is a list of five elements: it picks out from the sentence a subject noun-phrase, a verb, an object noun-phrase, and prepositional phrases of place and time, and lists them in that order. When one of the last three is missing, it writes (S) in its place. order works through the subfunctions picks, pickv, picko, pickpl, pickti. All of these take the same argument as order.

picks has
picks has as value the first noun-phrase before the verb, i.e. It looks in the list for an element beginning with N, NPL, or NTI and its value is the rest of that element. If it hits the verb before having found a suitable noun-phrase, its value is ERROR.

pickv looks in the list for an element beginning with V, or AUX1. Its value is the rest of the first such element it finds. If it finds none such, its value is ERROR.

picko looks for a verb followed immediately by an element tagged N, NTI or NPL. If the element following the verb is so marked, the value of picko is the rest of that element. If not, its value is (S).

pickpl looks for an element marked PNPL. If it finds one, pickpl1 looks for more. If more than one is found, they are consolidated to form the value of pickpl; if none is found, the value is (S).

pickti and picktil do exactly the same with PNTI.

Example:

X = ((PNTI AT NOON) (N JOHNNY) (V GOES) (PNPL UP THE STAIRS) (PNPL TO HIS ROOM))

order[X] = ((JOHNNY) (GOES) (S) (UP THE STAIRS TO HIS ROOM) (AT NOON))

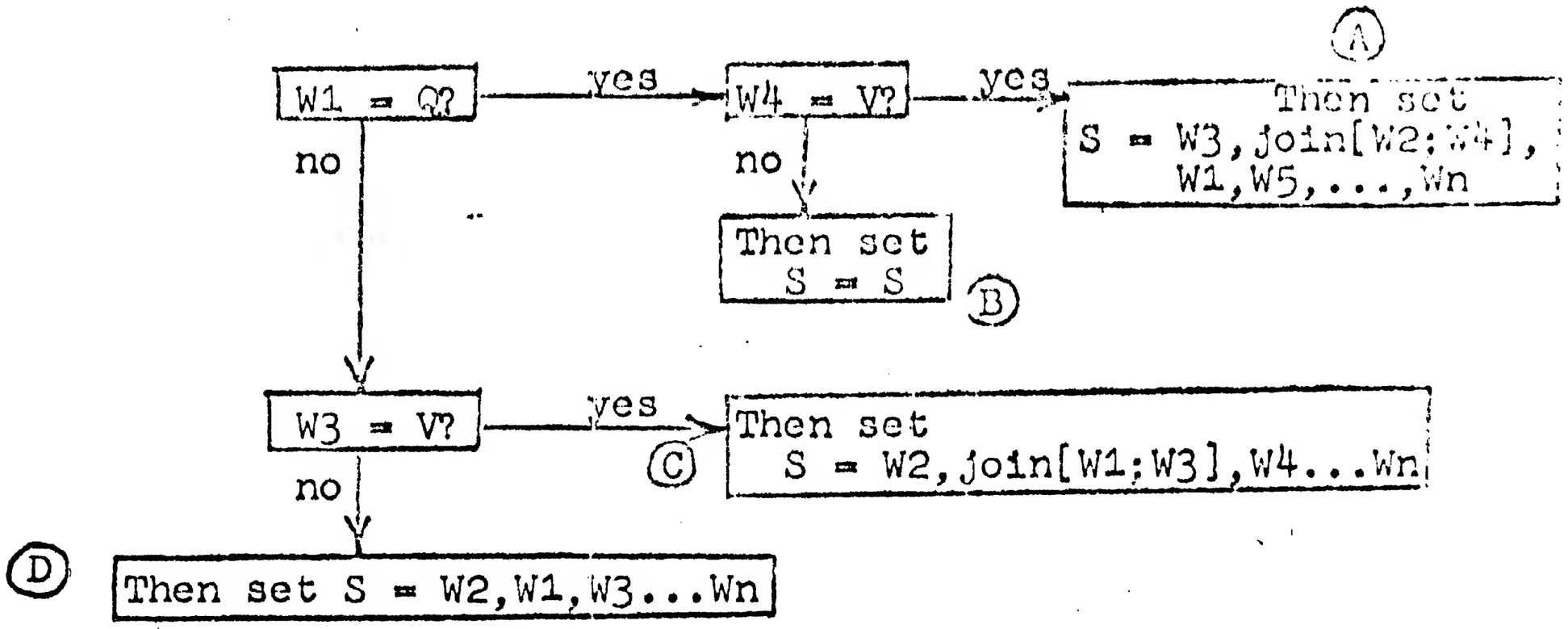
LISP definitions:

```
picks[X] = [null[X] → ERROR;member[caar[X];quote[(N NPL NTI)]] →
           cdar[X]; member[caar[X];quote[(V AUX1)]] → ERROR;
           T → picks[cdr[X]]]

pickv[X] = [null[X] → ERROR;member[caar[X];quote[(V AUX1)]] →
           cdar[X];T → pickv[cdr[X]]]

picko[X] = [member[caar[X];quote[(V AUX1)]] → [member[caadr[X];
           quote[(N NPL NTI)]] → cdadr[X];T → quote[(S)];
           T → picko[cdr[X]]]

pickpl[X] = [null[X] → quote[(S)];caar[X] = quote[PNPL] →
            append[cdar[X];pickpl1[cdr[X]]];T → pickpl[cdr[X]]]

pickpl1[X] = [null[X] → NIL;caar[X] = quote[PNPL] → append[cdar[X];
             pickpl1[cdr[X]]];T → pickpl1[cdr[X]]]

pickti[X] = [null[X] → quote[(S)];caar[X] = quote[PNTI] →
           append[cdar[X];pickti1[cdr[X]]];T → pickti[cdr[X]]]

pickti1[X] = [null[X] → NIL;caar[X] = quote[PNTI] → append[cdar[X];
             pickti1[cdr[X]]];T → pickti1[cdr[X]]]

order[X] = list[picks[X];pickv[X];picko[X];pickpl[X];pickti[X]]
```

revert[X] is a function of a list of pairs*, the result of applying lex and parse to the question. Its value is a list of pairs*. It rearranges the "parsed" question to permit matching with the text: in particular it undoes the inversion typical of the interrogative sentence. It has a subfunction join which is responsible for grouping the auxiliary and the verb when they have been separated by the interrogative construction, and join has a subfunction compare which deals especially with to do, for which it needs a table to tell it DOES + HAVE = HAS; DID + GO = WENT, etc. revert works as follows: (In the following diagram, S = W1,W2,...,Wn represents the sentence, and "W1 = Z?" means: "Is the tag on the first word or group of words (as formed by parse) Z?")

---

*Actually a list of lists. We consider each list as a pair whose first element is that of the list, and whose second element is the rest of the list.



Examples:

X = ((Q, WHERE) (AUX1 DOES) (N THE MAN) (V GO)) (A)

table contains ((DOES GO) GOES)
revert[X] = ((N THE MAN) (V GOES) (Q WHERE))

X = ((Q WHO) (AUX1 DOES) (N THE WORK)) (B)
revert[X] = ((Q WHO) (AUX1 DOES) (N THE WORK))

X = ((V HAS) (N THE PROFESSOR) (V GONE) (PNPL HOME)) (C)
revert[X] = ((N THE PROFESSOR) (V HAS GONE) (PNPL HOME))

X = ((V HAS) (N JOHN) (N A LIZARD)) (D)
revert[X] = ((N JOHN) (V HAS) (N A LIZARD))

join[X;Y] is a function of two pairs. Its value is a list formed from the first element of the second pair, and either the second elements of both pairs or the value of compare of the second elements, when it has to do with to do.

compare[X;Y;Z] is a function whose first two arguments are pairs, and whose third is a list of pairs of the form ((DOES GO) GOES), ((DID MEET) MET), etc. It forms the list of the last halves of its first two arguments and tries to match this with the first half of one of the elements of its third argument. Its value is then a pair made up from the second half of that element, preceded by the first half of the second argument. Its action is simple and easy to understand in an example. (Its value is ERROR if no match is found.)

Example:

X = (AUX1 DOES)
Y = (V GO)
Z contains ((DOES GO) GOES)
compare[X;Y;Z] = (V GOES)

LISP definitions:

```
revert[X] = [caar[X] = quote[Q] → [caadddr[X] = quote[V] →
                append[list[caddr[X];join[cadr[X];cadddr[X]]];
            cddddr[X]];T →(B) X];T → [caaddr[X] = quote[V] →(C)
                append[list[cadr[X];join[car[X];caddr[X]]];
            cdddr[X]];T →(D) append[list[cadr[X];car[X]];
            cddr[X]]]]

    join[X;Y] = [car[X] = quote[AUX1] → compare[X;Y;Z];T →
                append[append[cons[car[Y];NIL];cdr[X]];
                cdr[Y]]]

    compare[X;Y;Z] = [null[Z] → ERROR;append[cdr[X];cdr[Y]] =
                        caar[Z] → append[cons[car[Y];NIL];cdar[Z]];
                        T → compare[X;Y;cdr[Z]]]
```

edit[X] has as argument and value a list of lists, the result of lex, parse and revert of the question. Its responsibility is to substitute for the question-words the symbol W with the appropriate tag.

Example:

X = ((Q WHO) (Q WHAT) (Q WHOM) (Q WHEN) (Q WHERE))
edit[X] = ((N W) (N W) (N W) (PNTI W) (PNPL W))

LISP definition:

```
edit[X] = [null[X] → NIL;member[cadar[X];quote[(WHO WHAT WHOM)]] →
            append[quote[((N W))];edit[cdr[X]]];cadar[X] =
            quote[WHERE] → append[quote[((PNPL W))];edit[cdr[X]]];
            cadar[X] = quote[WHEN] → append[quote[((PNTI W))];
            edit[cdr[X]]];T → cons[car[X];edit[cdr[X]]]]
```

match[X;Y] is a function of two lists of lists. Its value is T (True) if the elements of each list in X are contained in the corresponding list in Y, and F (False) otherwise. match has a subfunction, contained which checks the individual lists one against the other, as follows:

contained[X;Y] is a function of two lists. Its value is T (True) if X is (S), if X is (W) and Y is not (S), or if each element of X appears in Y. Its value is F (False) otherwise. contained uses member which was defined to be the built-in function membob.

Examples:

X = (THE OLD MAN)
Y = (THE OLD MAN IN THE MOON)
contained[X;Y] = T

X = (THE MAN)
Y = (THE OLD WOMAN)
contained[X;Y] = F
(Note also that X = (THE OLD OLD MAN)
Y = (THE OLD MAN)
contained[X;Y] = T)

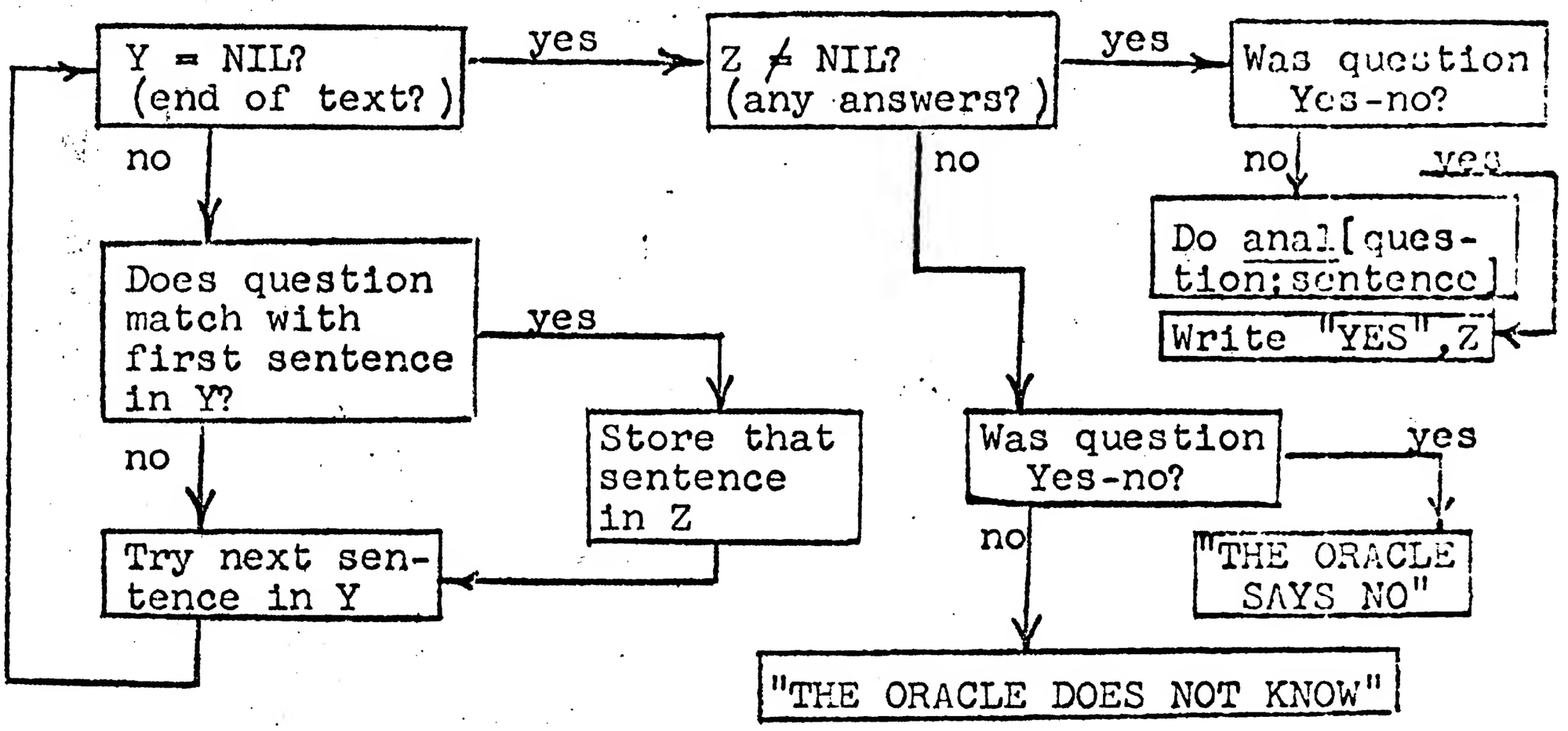
X = ((JOHNNY) (COMES) (S) (HOME) (S))
Y = ((JOHNNY APPLESEED) (COMES) (S) (HOME) (AT NOON))
match[X;Y] = T

X = ((JOHNNY) (COMES) (HOME))
Y = ((MARY) (COMES) (HOME))
match[X;Y] = F

LISP definitions:

```
contained[X;Y] = [null[X] → T;car[X] = quote[S] → T;
                  car[X] = quote[W]∧~car[Y] = quote[S] → T;
                  member[car[X];Y]∧contained[cdr[X];Y] → T;T → F]

match[X;Y] = [null[X] → T;contained[car[X];car[Y]]match[cdr[X];
              cdr[Y]] → T;T → F]
```

answer[X;Y;Z] is a function of three arguments: X is the processed question; Y is a list of processed sentences; Z accumulates the sentences of Y which have matched successfully with the question X. Its value is an appropriate answer to the question. answer operates as follows:



anal is a subfunction of answer.

anal]X;Z] is a function of two arguments: the first is a list, the question; the second is a list of lists, the answers. The responsibility of anal is to pick out the words in the answers which matched with the question-words (who, where, etc.) in the question. The actual searching is done by ques.

ques[X;Z] has as its arguments two lists: X is the question, Y is one of the answers. ques searches through the question for "W"'s and its value is the list of the elements of the answer that correspond to "W"'s in the question.

Examples:

X = ((W) (COMES) (W))
Y = ((JOHNNY) (COMES) (HOME))
ques[X;Y] = ((JOHNNY) (HOME))

X = ((W) (COMES) (S))
Y = (((JOHNNY) (COMES) (HOME)) ((MARY) (COMES) (AWAY)))
anal[X;Y] = (((JOHNNY) ((JOHNNY) (COMES) (HOME)))
((MARY) ((MARY) (COMES) (AWAY))))

Note that in the description below ques is also used as a subfunction of anal, to determine whether or not we have a yes-no question. This is done by null[ques[X;quote[(B B B B B)]]] which has value T only if there were no "W"'s in X.

LISP definitions:

```
answer[X;Y;Z] = [ null[Y] → [null[Z] → [null[ques[x;
                quote[(B B B B B)]]] → quote[(THE ORACLE SAYSNNO)];
                T → quote[( THE ORACLE DOES NOT KNOW)]];
                T → [null[ques[X;quote[(B B B B B)]]] →
                list[quote[YES];Z];T → anal[X;Z]]];T →
                [match[X;car[Y]] → answer[X;cdr[Y];append[Z;
                cons[car[Y];NIL]]];T → answer[X;cdr[Y];Z]]]

    anal[X;Z] = [null[Z] → NIL;T → append[cons[append[ques[X:
                car[Z]];cons[car[Z];NIL]];NIL];anal[X;cdr[Z]]]]

    ques[X;Y] = [null[X] → NIL;caar[X] = quote[W] → cons[car[Y];
                ques[cdr[X];cdr[Y]]];T → ques[cdr[X];cdr[Y]]]
```

textanal[X] processes the list of sentences:

```
    textanal[X] = [null[X] → NIL;T → append[cons[order[parse[
                  lex[car[X];GRAMM1];GRAMM2]];NIL];textanal[cdr[X]]]]
```

oracle[X;Y] has as value the list of answers from sentences in Y to the question X: oracle[X;Y]

```
    oracle[X;Y] = [fix[answer[order[edit[revert[parse[lex[X];
                  GRAMM1];GRAMM2]]]];textanal[Y];NIL]]
```

The values of answer are of the form:

(((IN THE CLASSROOM) ((THE TEACHER) (READS) (BOOKS) (IN THE CLASSROOM) (S))))

(YES (((THE TEACHER) (READS) (BOOKS) (IN THE CLASSROOM) (S))))

(THE ORACLE SAYS NO)

The function fix, with its subfunctions unparen and unp has the responsibility for rewriting these as:

(((IN THE CLASSROOM) (THE TEACHER READS BOOKS IN THE CLASSROOM)))
(YES ((THE TEACHER READS BOOKS IN THE CLASSROOM)))
(THE ORACLE SAYS NO)

```
fix[X] = [null[X] → NIL;atom[car[X]] → cons[car[X];
          fix[cdr[X]]];T → cons[unparen[car[X]];fix[cdr[X]]]]

unparen[X] = [null[X] → NIL;T → cons[unp[car[X]];unparen[cdr[X]]]]

unp[X] = [null[X] → NIL;atom[car[X]] → X;car[X] = quote[(S)] →
          unp[cdr[X]];T → append[car[X];unp[cdr[X]]]]
```

Conclusions

Conclusions.

To achieve our original goal of passing a six-year-old's reading-comprehension test we would need, as mentioned in the introduction, a complex syntax-recognizer, a logic machine and an encyclopedia. Syntax recognition by machine will not come tomorrow, but it is on its way. Yngve's hypotheses (see Bibliography) seem both revealing of the true nature of language and especially appropriate for machine use. Theorem provers have been built, and perhaps a program could be written to simulate the logical processes of a child. The encyclopedia does not seem entirely out of reach, either. With lists of synonyms, and tabulation of objects into various categories, e.g. animate and inanimate, and perhaps just stockpiling information about every-day life, one might be able to reconstruct the world as a six-year old knows it, the context in which he passes his reading-comprehension tests. Certainly programming languages of high sophistication, such as LISP, used here, or COMIT*, designed for convenient manipulation of symbolic elements, will be an essential part of such an undertaking. From here, it looks feasible.

---

*See Yngve, "A Programming Language for Mechanical Translation", Mechanical Translation, Vol. 5, No. 1, July, 1958.

Bibliography

Psychology

1. Piaget and Inhelder, La Genese des Structures Logiques Elementaires, Delachaux et Niestle, Neuchatel, 1959.

Syntactic Analysis

1. Chomsky, N. A., Syntactic Structures, Mouton and Co., 'S-Gravenhage, 1957.

2. Yngve, V. H., A Model and an Hypothesis for Language Structure, in press.

Machine Codes

1. McCarthy, John, Recursive Functions of Symbolic Expressions and their Computation by Machine, Communications of the ACM, Vol. 3, No. 4, April, 1960.

2. Yngve, V. H., A Programming Language for Mechanical Translation, Mechanical Translation, Vol. 5, No. 1, July, 1958.

CS-TR Scanning Project
Document Control Form

Date : 11/30/95

Report # AIM - 16

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☐ Technical Report (TR) ☒ Technical Memo (TM)

☐ Other:_____

Document Information

Number of pages: 20 (24-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:

☐ Single-sided or

☒ Double-sided

Intended to be printed as :

☐ Single-sided or

☒ Double-sided

Print type:

☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☒ Other: COPY OF MIMEOGRAPH

Check each if included with document:

☐ DOD Form ☐ Funding Agent Form ☐ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number): _____

Photographs/Tonal Material (by page number): _____

Other (note description/page number):

Description :	Page Number:
IMAGE MAP: (1-20)	1-20
(21-24) SCANCONTROL TRGT	

Scanning Agent Signoff:

Date Received: 11/30/95 Date Scanned: 12/13/95 Date Returned: 12/14/95

Scanning Agent Signature: Michael W. Cook

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**